THÉORIE DES TYPES

Patrick Bellot

Télécom Paris
Institut Polytechnique de Paris
bellot@telecom-paris.fr

MITRO 202

Citons :

INTRODUCTION TO COMBINATORS AND λ-CALCULUS

Roger Hindley & Jonathan P. Seldin
London Mathematical Society, Student Text I, Cambridge Univ. Press, London, 1986

et

LOGIC: FORM AND FUNCTIONS

J.A. Robinson
Edinburgh University Press, Edinburgh, Écosse, 1979

Introduction

Les combinateurs ou les λ-termes ne sont pas des fonctions des mathématiques classiques, ne serait-ce que parce que l'on autorise l'auto-application qui est interdite par la théorie des ensembles. Les combinateurs et les λ-termes n'ont ni ensemble de départ ni ensemble d'arrivée. Ils peuvent prendre n'importe quoi en argument et il est possible que leurs résultats soient indéfinis.

Ce chapitre présente un point de vue différent, principalement dû au Paradoxe de Curry. Il s'agit de l'introduction de types qui permettent ou ne permettent pas certaines applications d'un terme à un autre et qui rapprochent le comportement des combinateurs de celui des fonctions de la théorie des ensemble.

La théorie des types est aussi appelée théorie de la *fonctionnalité*.

Le paradoxe de Curry

L'idée de départ de certains logiciens était de mécaniser entièrement la logique, d'en faire un calcul comme l'arithmétique. Cela remonte aux année 1600. C'est aussi pourquoi la théorie des combinateurs portait le nom de Logique Combinatoire car c'était aussi l'intention première du logicien H.B. Curry au début du XX$^{\text{ème}}$ siècle.

Pour cela, l'approche de Curry consistait à supprimer les variables liées dans les formules grâce à l'utilisation des combinateurs comme nous l'avons déjà vu avec le théorème d'abstraction.

Il faut bien comprendre que les variables sont un outil particulièrement complexe et que la définition de la substitution $[t/x]A$ d'un terme t à une variable x dans une formule A fut longtemps mal définie et que des logiciens parmi les meilleurs se trompèrent à essayer de la définir.

Le paradoxe de Curry

Curry introduit des combinateurs, constantes atomiques, pour représenter les différents connecteurs et quantificateurs de la logique du premier ordre. Il introduit en particulier P tel que $(P\ A\ B)$ représente l'implication logique $A \Rightarrow B$.

Curry donne la règle d'égalité:

Si A est vrai et si $A = B$ alors B est vrai

où $=$ est l'égalité de la théorie des combinateurs.

Avec cette règle, on pourra déduire une proposition d'une autre par un *simple calcul* dans la théorie des combinateurs.

L'idée était belle !

Le paradoxe de Curry

Puis Curry introduit les deux règles essentielles suivantes:

De $(P\ X\ Y)$ et X déduire Y

et:

$$P\ (P\ A\ (P\ A\ B))\ (P\ A\ B)$$

Ces deux règles sont des règles de la logique propositionnelle toute simple. C'est un exemple de l'une des nombreuses axiomatisations possibles de la logique propositionnelle. N'importe quelle autre axiomatisation permettrait de déduire cet axiome pour toutes les formules A et B comme un théorème. Même en se restreignant à ce cadre tout simple, un paradoxe apparaît, c'est le paradoxe de Curry.

Le paradoxe de Curry

Soit Z une proposition, nous allons démontrer que Z est un théorème. Considérons le terme:

$$F_Z \equiv_{def} Y\ (\lambda^+ z\ \cdot\ P\ z\ (P\ z\ Z))$$

où $z \notin FV(Z)$ et où Y est l'opérateur de point-fixe que nous avons déjà vu. On a :

$$\begin{array}{rcll} F_Z & = & (\lambda^+ z\ \cdot\ P\ z\ (P\ z\ Z))\ F_Z & (\text{propriété du point-fixeur } Y) \\ & = & P\ F_Z\ (P\ F_Z\ Z) & (\text{substitution}) \\ & = & P\ (P\ F_Z\ (P\ F_Z\ Z))\ (P\ F_Z\ Z) & (\text{remplacement du } 1^{er} F_Z \text{ par sa valeur}) \end{array}$$

On effectue à présent une déduction:

$$\begin{array}{rcll} & & P\ (P\ F_Z\ (P\ F_Z\ Z))\ (P\ F_Z\ Z) & (\text{axiome}) \\ & = & P\ F_Z\ (P\ F_Z\ Z) & (\text{règle d'égalité}) \\ & = & F_Z & (\text{règle d'égalité*}) \\ & = & P\ F_Z\ Z & (\text{modus ponens}) \\ & = & Z & (\text{modus ponens}) \end{array}$$

On en arrive donc à l'absurdité suivante qui constitue le paradoxe de Curry : toute proposition Z est vraie.

Le paradoxe de Curry

Puisque toute proposition Z est un théorème, la théorie obtenue par Curry est *inconsistante*.

On ne peut remettre en cause l'algorithme λ^+, *i.e. théorème de complétude*, ni l'existence de l'opérateur de point-fixe Y.

Si l'on veut continuer dans cette voie, il faut autre chose.

La solution trouvée, qui conduit naturellement à la théorie des types, consiste à dire que tous les termes ne représentent pas des propositions.

La théorie des types est constituée de règles permettant d'autoriser ou d'interdire l'application d'un terme à un autre et donc d'interdire certains termes.

Théorie des types vs. théorie des ensembles

Un examen attentif des paradoxes montre qu'ils semblent avoir tous la même source : l'auto-référence, que ce soit le paradoxe du Crétois qui dit *Le Crétois dit qu'il ment*, que ce soit le paradoxe de Russel *L'ensemble de tous les ensembles qui ne se contiennent pas eux-mêmes*$^{(*)}$ ou le paradoxe de Curry que nous venons de voir.

Dans le premier cas, le Crétois parle aussi de lui-même. Dans le deuxième cas, la définition de l'ensemble fait référence à lui-même. Dans le troisième cas, c'est l'opérateur de point-fixe Y qui provoque une auto-référence.

(*) si $R = \{x \mid x \notin x\}$ alors $R \in R \Leftrightarrow R \notin R$.

Théorie des types vs. théorie des ensembles

Les origines des paradoxes se trouvent dans la conjonction de ces deux principes :

- tout prédicat peut être considéré pour définir un ensemble ;
- un prédicat peut être appliqué à n'importe quel argument.

Il y a alors deux solutions pour empêcher le paradoxe d'apparaître.

Théorie des ensembles

L'idée original (et naïve) de Cantor et Frege a été d'introduire la notation $\{x \mid P(x)\}$ pour définir un ensemble par sa propriété caractéristique P.

La première possibilité est la solution *ensembliste* qui consiste à dire que tout prédicat ne définit pas un ensemble.

On différenciera donc les prédicats bien définis qui définissent des ensembles et les autres.

Par exemple, le prédicat $x \notin x$ ne définit pas un ensemble, parce qu'il n'est pas *bien* défini, de telle sorte que l'ensemble de Russel n'est pas un ensemble mais une vague collection que l'on appelle une *classe* et pour laquelle le prédicat d'appartenance ne s'utilise pas.

La théorie *moderne* des ensembles (ZF = Zermelo-Frankel) choisit d'abandonner le premier des deux principes.

Théorie des types

La deuxième solution est celle de la *théorie des types* qui choisit d'abandonner le deuxième principe.

On empêche arbitrairement, ou presque, certaines applications de fonctions ou de prédicats.

Dans notre cas, Y n'aura pas de type.

Nous décrirons la théorie des types dans le cas de la théorie des combinateurs. Elle s'adapte aussi au λ-calcul.

La théorie des types que nous présentons a été développée par Curry en 1934 et 1936. Elle fut développée indépendamment par J.H. Morris et R. Milner (le langage ML) dans le cadre des langages de programmation fonctionnelle dans les années 1970.

Les types en informatique

Le plus simple des contrôles de type est celui qui contrôle uniquement le nombre des arguments.

Le pas suivant est franchi lorsque l'on contrôle en plus la nature des arguments, ce que font la plupart des langages de programmation où le programmeur déclarent les types de objets (fonctions et données) qu'il manipule.

Enfin, certains langages fonctionnels comme ML infèrent le type des objets qu'ils manipulent.

Le résultat, comme nous allons le voir est un *schéma de types*.

L'approche de Curry

Pour Curry, les types des termes sont inférés grâce à un ensemble de règles.

Chaque terme possède un ensemble de types possibles et non un seul.

Si un terme M possède un type α, on notera cela : $\vdash M : \alpha$.

Parmi tous les types que possède un terme, lorsqu'il en possède, l'un d'entre eux est le plus général en ce sens que les autres peuvent se déduire du type principal en substituant d'autre schémas de types aux variables de types qu'il contient.

Définition des types

On se donne un ensemble de constantes de types et un ensemble de variables de types qui sont notées à l'aide des lettres grecques minuscules α, β, γ, ... éventuellement indicées par des entiers naturels.

Les *schémas de types* sont définis par induction :

- toute constante de type est un schéma de type ;
- toute variable de type est un schéma de type ;
- si α et β sont des schémas de types, $(\alpha \to \beta)$ est un schéma de types ;
- *règle de fermeture*.

Définition des types

Un *type* est un schéma de types dans lequel ne figure aucune variable.

Un schéma de types qui contient le symbole $\rightarrow$ est dit *complexe*.

On suppose que l'opération $\rightarrow$ de formation de types est associative à droite. Donc $\alpha \rightarrow \beta \rightarrow \gamma$ et $\alpha \rightarrow (\beta \rightarrow \gamma)$ dénotent la même expression.

Affectation de types

L'idée principale est de dire que si un terme M a le type $\alpha \to \beta$ et si N a le type α alors $(M\ N)$ est bien défini et a le type β.

L'affectation de type est présentée sous la forme d'un système formel où les formules démontrées sont de la forme $\vdash M : \alpha$ où M est un terme de la Logique Combinatoire et α un schéma de types.

Ce système possède une seule règle d'inférence donnant le type d'une application et deux axiomes donnant les types des combinateurs S et K.

Type d'une application

Cette règle reprend ce qui a été dit ci-dessus:

$$\frac{\vdash M : \alpha \to \beta \quad \vdash N : \alpha}{\vdash (M\ N) : \beta}$$

Le type de K

La première règle donne un schéma de types à K :

$$\vdash K : \alpha \to \beta \to \alpha$$

Ce schéma de types signifie informellement que K prend un premier argument X de type α, un deuxième argument Y de type β et l'application se réduit à un terme de type α, ce qui est normal puisque la règle de réduction de K est $K\ X\ Y\ :=\ X$.

α et β sont des variables de types. On peut leur substituer n'importe quel schéma de types pour obtenir un nouveau schéma de types pour K. Ainsi , le schéma de types $(\gamma \to \gamma) \to \beta \to \gamma \to \gamma$ est aussi un schéma de types pour K obtenu en remplaçant α par $\gamma \to \gamma$.

Le type de S

La deuxième règle donne un schéma de types à S :

$$\vdash S : (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$$

En raisonnant comme pour K, supposons que:

$$\vdash X : \alpha \to \beta \to \gamma \qquad \vdash Y : \alpha \to \beta \qquad \vdash Z : \alpha$$

on peut alors intuitivement déduire successivement:

$$\vdash X\ Z : \beta \to \gamma \qquad \vdash Y\ Z : \beta$$

puis enfin:

$$\vdash X\ Z\ (Y\ Z) : \gamma$$

Comme pour le schéma de types de K, on peut substituer des schémas de types aux variables α, β et γ et obtenir un autre schéma de types pour S.

Schéma de types d'une application

Propriété. Soient M et N deux termes, soit α un schéma de types tel que $\vdash (M\ N) : \alpha$ alors il existe un schéma de type β tel que $\vdash M : \beta \to \alpha$ et $\vdash N : \beta$.

La démonstration se fait en examinant la déduction qui a permis de conclure $\vdash (M\ N) : \alpha$. La dernière règle utilisée est forcément celle qui donne le type d'une application. En effet, les deux autres règles donnent les types de termes atomiques. La dernière étape de la démonstration est donc de la forme:

$$\frac{\vdash M : \beta \to \alpha \quad \vdash N : \beta}{\vdash (M\ N) : \alpha}$$

On retrouve ainsi β.

Déduction d'un schéma de types de I

$$\dfrac{\dfrac{\vdash S : (\rho \to (\rho \to \rho) \to \rho) \to (\rho \to \rho \to \rho) \to \rho \to \rho \qquad \vdash K : \rho \to (\rho \to \rho) \to \rho}{\vdash (S\ K) : (\rho \to \rho \to \rho) \to \rho \to \rho} \qquad \vdash K : \rho \to \rho \to \rho}{\vdash (S\ K\ K) : \rho \to \rho}$$

type-schéma de S avec
$\alpha \equiv \rho$
$\beta \equiv \rho \to \rho$
$\gamma \equiv \rho$

type-schéma de K avec
$\alpha \equiv \rho$
$\beta \equiv \rho \to \rho$

type-schéma de K avec
$\alpha \equiv \rho$
$\beta \equiv \rho$

$$\dfrac{\vdash (S\ K\ K) : \rho \to \rho}{\vdash I : \rho \to \rho}$$

Il est rassurant de constater que $\rho \to \rho$ est un schéma de types pour I puisque sa règle de réduction est $I\ X\ :=\ X$.

Exercices

1) Commencer par donner intuitivement les schémas de types des combinateurs B, C, C_* et W.

2) Puis vérifier qu'on les obtient bien à l'aide du système formel d'affectation de types donnés ci-dessus.

3) Vous convaincre intuitivement que le combinateur $(\lambda^+ x \cdot xx)$ n'a pas de type.

Les contextes

Pour l'instant, les seuls termes que nous typons sont des termes ne contenant pas de variable. On peut typer un terme contenant des variables à condition de supposer des types pour ces variables.

Une *base de typage* ou *contexte* est un suite de formules:

$$x_1 : \alpha_1, \ldots, x_n : \alpha_n, \ldots$$

où $\{x_1, \ldots, x_n, \ldots\}$ est un ensemble qui peut être infini de variables deux à deux différentes et où $\{\alpha_1, \ldots, \alpha_n, \ldots\}$ est un ensemble de schémas de types.

Nouvelles formules

On choisit de démontrer à présent des formules de la forme:

$$x_1 : \alpha_1, \ldots, x_n : \alpha_n, \ldots \vdash M : \alpha$$

signifiant que M a le type α sous des hypothèses concernant les types des variables $x_1 : \alpha_1, \ldots, x_n : \alpha_n, \ldots$

Nouveaux axiomes

On ajoute l'axiome suivant :

$$x_1 : \alpha_1, \ldots, x_n : \alpha_n, \ldots \vdash x_i : \alpha_i$$

On réécrit les trois règles déjà vues en ajoutant une base représentée par le symbole $\mathcal{B}$ à gauche du symbole $\vdash$ dans chacune des règles déjà présentées. Soit :

$$\frac{\mathcal{B} \vdash M : \alpha \to \beta \quad \mathcal{B} \vdash N : \alpha}{\mathcal{B} \vdash (M\ N) : \beta}$$

$$\mathcal{B} \vdash K : \alpha \to \beta \to \alpha$$

$$\mathcal{B} \vdash S : (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$$

Exercice

Démontrer:

$$f : \beta_1 \to \beta_2 \to \gamma, g : \alpha \to \beta_1, h : \alpha \to \beta_2, x : \alpha \vdash f\ (g\ x)\ (h\ x) : \gamma$$

Type d'une application

Propriété. Soit $\mathcal{B}$ une base de typage, soient M et N deux termes, soit α un schéma de types tel que $\mathcal{B} \vdash (M\ N) : \alpha$ alors il existe un schéma de types β tel que $\mathcal{B} \vdash M : \beta \to \alpha$ et $\mathcal{B} \vdash N : \beta$.

La même que précédemment.

Abstraction et typage

Propriété. Soit $x : \alpha, \ldots$ une base de typage, soit β un schéma de types et M un terme tel que : $x : \alpha, \ldots \vdash M : \beta$, alors on a: $\ldots \vdash (\lambda^+ x \cdot M) : \alpha \to \beta$.

La démonstration se fait classiquement par induction sur la structure du terme M :

- Si $M \equiv x$, alors $\beta \equiv \alpha$ et l'on a bien $(\lambda^+ x \cdot x) \equiv I$ dont le schéma de types est $\alpha \to \alpha$.
- Si $M \equiv y$ une variable différente de x ou une constante, alors $(\lambda^+ x \cdot y) \equiv K\ y$ est bien de type $\alpha \to \beta$.
- Si $M \equiv P\ Q$ alors on a : $\vdash P : \gamma \to \beta$ et $\vdash Q : \gamma$. Par hypothèse d'induction, on a : $\vdash \lambda^+ x \cdot P : \alpha \to \gamma \to \beta$ et $\vdash \lambda^+ x \cdot Q : \alpha \to \gamma$. On vérifie alors que : $(\lambda^+ x \cdot M) \equiv S\ (\lambda^+ x \cdot P)\ (\lambda^+ x \cdot Q)$ est bien de type $\alpha \to \beta$.

Réduction et typage

Propriété. Soit $\mathcal{B}$ une base de typage, soit M et N des termes tels que $M := N$, soit α un schéma de types tels que $\mathcal{B} \vdash M : \alpha$, alors on a: $\mathcal{B} \vdash N : \alpha$.

La démonstration se fait par récurrence sur la hauteur de la preuve de $M := N$. Il faut démontrer que la propriété est vraie pour chacun des axiomes. Puis pour chacune des règles d'inférences, il faut démontrer que si la propriété est vraie pour les prémisses de la règle, elle est aussi vraie pour la conclusion de la règle. La démonstration ne pose pas de difficulté particulière et se sert de la propriété *Type d'une application*.

Réduction et typage

On n'a pas le résultat inverse du théorème ci-dessus, ce n'est pas parce que N a un schéma de types que M a le même.

Prenons $M \equiv S\ K\ S\ I$ et $N \equiv K\ I\ (S\ I)$. On a $M := N$. On a $\vdash N : \alpha \to \alpha$ mais on n'a pas $\vdash M : \alpha \to \alpha$. On peut démontrer que le schéma de types principal de $(S\ K\ S\ I)$ est $(\alpha \to \beta) \to (\alpha \to \beta)$.

Un exemple encore plus significatif est $M \equiv S\ I\ I\ I$ et $N \equiv I\ I\ (I\ I)$. On a aussi $M := N$. On a $\vdash N : \beta \to \beta$ mais M n'a pas de type car $(S\ I\ I)$ n'a pas de type.

Substitution et typage

Propriété. Soit $x : \alpha, \ldots$ une base de typage, soit β un schéma de types et M un terme tel que: $x : \alpha, \ldots \vdash M : \beta$, alors si $\ldots \vdash N : \alpha$, on a $\ldots \vdash [N/x]M : \beta$.

La démonstration se fait par induction sur la structure du terme M. Si $M \equiv x$, le résultat est évident. Si $M \equiv y$, une variable différente de x, le résultat est évident. Si $M \equiv (P\ Q)$, on s'appuie sur l'hypothèse d'induction et sur la propriété *Type d'une application*.

Schéma de types principal

Soient α et β deux schémas de types, on dit que β est une instance de α s'il existe des variables de type $\alpha_1, \ldots, \alpha_n$ et des schémas de types $\beta_1, \ldots, \beta_n$ tels que $\beta \equiv [\beta_1/\alpha_1, \ldots, \beta_n/\alpha_n]\alpha$.

Ainsi, par exemple $(\alpha \to \beta) \to (\alpha \to \beta)$ est une instance de $\alpha \to \alpha$.

Soient M un terme, soit α un schéma de types, $\mathcal{B}$ une base de typage, on dit que α est le schéma de type principal de M si pour tout schéma de types β tel que $\mathcal{B} \vdash M : \beta$ on a que β est une instance de α.

Schéma de types principal

Théorème. Tout terme possédant un schéma de types pour une base de typage possède un schéma de types principal pour cette base.

Ce schéma de types principal est unique au renommage des variables près.

La démonstration se trouve dans (Curry & Hindley 1969).

Typage et normalisation forte

Théorème. Tout terme possédant un schéma de types est fortement normalisable, i.e. il est normalisable et toutes les stratégies de réduction le normalisent.

Autrement dit, l'arbre de réduction d'un terme typable est fini.

Preuve dans (Curry et al. 1958).

Expressivité de la théorie typée des combinateurs

On en s'occupe que des fonctions sur les nombres naturels, les entiers de Church.

On appelle fonctions polynomiales étendues les fonctions que l'on peut obtenir à partir de $\underline{0}$ (zéro), $\underline{s}$ (fonction successeur), $\underline{z}$ (test d'égalité à zéro), $\underline{+}$ (addition) et $\underline{*}$ (produit) par composition.

Théorème de Schwichtenberg. Les fonctions sur les entiers de Church exprimables en théorie typée des combinateurs sont les fonctions polynomiales étendues.

Algorithme de Morris

L'algorithme de Morris permet de calculer le type d'un terme.

Si le terme a un type, l'algorithme trouve le type principal.

Si le terme n'a pas de type, l'algorithme échoue.

Il s'agit de trouver un schéma de types α tel que $\mathcal{B} \vdash M : \alpha$.

L'algorithme s'exprime par induction sur M.

Algorithme de Morris

Si le terme est S, son schéma de types est donné par l'axiome correspondant dans lequel on renommera les variables de types avec de nouvelles variables.

$$\vdash S : (\alpha_i \to \beta_i \to \gamma_i) \to (\alpha_i \to \beta_i) \to \alpha_i \to \gamma_i$$

Si le terme est S, son schéma de types est donné par l'axiome correspondant dans lequel on renommera les variables de types avec de nouvelles variables.

$$\vdash K : \alpha_j \to \beta_j \to \alpha_j$$

Si le terme est une variable, son schéma de types est donné par la base de typage. Si son schéma de types n'est pas donné par la base de typage, le typage échoue.

Algorithme de Morris

Si le terme est de la forme $(M\ N)$, on calcule récursivement α schéma de types principal de M et β schéma de type principal de N. Si l'un des deux calculs échoue, $(M\ N)$ n'a pas de type.

Si α est une variable de type, on introduit une nouvelle variable de type γ, on remplace partout α par $\beta \to \gamma$ et le schéma de types principal de $(M\ N)$ est γ.

Si α et un type complexe $\delta \to \gamma$, on pose l'équation $\beta = \delta$. Cette équation va être résolue. La résolution peut échouer et dans ce cas, le typage échoue. La résolution peut réussir et dans ce cas, elle remplace *partout* des variables de types par des schémas de types. Le schéma de types principal de $(M\ N)$ est alors γ dans lequel des variables de types auront été remplacées.

Algorithme de Morris

La résolution de $\beta = \delta$ se décrit ainsi :

- si β est une variable de type et si β apparaît dans δ, il n'y a pas de solution et la résolution échoue ;
- sinon, si β est une variable de type et si β n'apparaît pas dans δ, on remplace *partout* β par δ ;
- sinon, si δ est une variable de type, on remplace l'équation par $\delta = \beta$ et on continue la résolution sur cette nouvelle équation ;
- sinon, on a $\beta \equiv \beta_1 \to \beta_2$ et $\delta \equiv \delta_1 \to \delta_2$, on commence par résoudre $\beta_1 = \delta_1$: si cette résolution échoue, alors toute la résolution échoue ; si cette résolution réussit, on résout ensuite $\beta_2 = \delta_2$, résolution qui peut réussir ou échouer.

Algorithme de Morris

Présenté ainsi, l'algorithme peut sembler complexe.

On peut le résumer ainsi. Si $\vdash M : \alpha \to \beta$ et $\vdash N : \gamma$, on pose l'équation $\alpha = \gamma$.

Si cette équation peut être résolue sans circularités, on remplace les variables par leurs valeurs dans β et on obtient le type de $(M\ N)$.

Exercice

Essayer de trouver le type de W tel que :

$$W\ f\ x\ =\ f\ x\ x$$

Puis calculer ce type avec l'algorithme de Morris sachant que

$$W\ \equiv_{def}\ S\ S\ (K\ I)$$

Mais, ça, vous devriez le savoir depuis le temps 🙂

Solution

Nous avons $W\ f\ x\ =\ f\ x\ x$.

Dans $f\ x\ x$, f est appliquée deux fois à x.

Si $\vdash x : \alpha$, alors $\vdash f : \alpha \to \alpha \to \beta$.

Et le résultat est de type β.

Donc, W prend un premier argument de type $\alpha \to \alpha \to \beta$, un deuxième argument de type α et le résultat est de type β.

Donc : $\vdash W : (\alpha \to \alpha \to \beta) \to \alpha \to \beta$.

Essayons de le vérifier avec l'algorithme de Morris.

Solution

On considère $W \equiv_{def} S\ S\ (K\ I) \equiv_{def} (S\ S)\ (K\ I)$.

Il faut donc calculer :

- le schéma de types de $(S\ S)$,
- puis le schéma de types de $(K\ I)$
- et vérifier que $(K\ I)$ peut être un argument pour $(S\ S)$.

Bien entendu, si l'une de ses étapes échouaient,
W n'aurait pas de type.

Solution

On calcule le type de $(S\ S)$:

- $\vdash S : (\alpha_1 \to \beta_1 \to \gamma_1) \to (\alpha_1 \to \beta_1) \to \alpha_1 \to \gamma_1$
- $\vdash S : (\alpha_2 \to \beta_2 \to \gamma_2) \to (\alpha_2 \to \beta_2) \to \alpha_2 \to \gamma_2$

Puis résolution de :
$\alpha_1 \to \beta_1 \to \gamma_1 = (\alpha_2 \to \beta_2 \to \gamma_2) \to (\alpha_2 \to \beta_2) \to \alpha_2 \to \gamma_2$.
qui donne :
$\alpha_1 = \alpha_2 \to \beta_2 \to \gamma_2$
$\beta_1 \to \gamma_1 = (\alpha_2 \to \beta_2) \to \alpha_2 \to \gamma_2$
puis :
$\alpha_1 = \alpha_2 \to \beta_2 \to \gamma_2$
$\beta_1 = \alpha_2 \to \beta_2$
$\gamma_1 = \alpha_2 \to \gamma_2$
Il n'y a aucune circularité. On remplace les 3 variables par leurs valeurs dans $(\alpha_1 \to \beta_1) \to \alpha_1 \to \gamma_1$ pour obtenir le schéma de types de $(S\ S)$:

$\vdash S\ S : ((\alpha_2 \to \beta_2 \to \gamma_2) \to \alpha_2 \to \beta_2) \to (\alpha_2 \to \beta_2 \to \gamma_2) \to \alpha_2 \to \gamma_2$

Solution

On calcule le type de $(K\ I)$:

- $\vdash K : \alpha_3 \to \beta_3 \to \alpha_3$
- $\vdash I : \alpha_4 \to \alpha_4$ (on l'avait calculé plus haut)

La résolution de $\alpha_3 = \alpha_4 \to \alpha_4$ est immédiate et fournit le schéma de types de $(K\ I)$:

$\vdash K\ I : \beta_3 \to (\alpha_4 \to \alpha_4)$.

Solution

$\vdash S\ S : ((\alpha_2 \to \beta_2 \to \gamma_2) \to \alpha_2 \to \beta_2) \to (\alpha_2 \to \beta_2 \to \gamma_2) \to \alpha_2 \to \gamma_2$
$\vdash K\ I : \beta_3 \to (\alpha_4 \to \alpha_4)$
Puis résolution de :
$(\alpha_2 \to \beta_2 \to \gamma_2) \to \alpha_2 \to \beta_2 = \beta_3 \to (\alpha_4 \to \alpha_4)$.
qui donne :
$\beta_3 = \alpha_2 \to \beta_2 \to \gamma_2$
$\alpha_4 \to \alpha_4 = \alpha_2 \to \beta_2$
Puis :
$\beta_3 = \alpha_2 \to \beta_2 \to \gamma_2$
$\alpha_4 = \alpha_2$
$\alpha_4 = \beta_2$
Après remplacement du premier α_4 par α_2 :
$\beta_3 = \alpha_2 \to \beta_2 \to \gamma_2$
$\alpha_2 = \beta_2$
Après remplacement de α_2 par sa valeur dans la première équation :
$\beta_3 = \beta_2 \to \beta_2 \to \gamma_2$
$\alpha_2 = \beta_2$
Comme il n'y a pas de circularité, on remplace β_3 et α_2 dans :
$(\alpha_2 \to \beta_2 \to \gamma_2) \to \alpha_2 \to \gamma_2$ pour obtenir le type de $S\ S\ (K\ I)$:

$$\vdash W\ : (\beta_2 \to \beta_2 \to \gamma_2) \to \beta_2 \to \gamma_2$$

Exercice

On vient de voir que :

$$\vdash W : (\alpha \to \alpha \to \beta) \to \alpha \to \beta$$

- Calculer $W\ I\ x$.
- Montrer que $W\ I$ n'a pas de type.

Exercice

On admet que :

$\vdash W : (\alpha \to \alpha \to \beta) \to \alpha \to \beta$
$\vdash B : (\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$

Montrer que l'opérateur de point-fixe
$Y \equiv_{def} W\ S\ (B\ W\ B)$
n'a pas de type.

Conclusions

Il y a beaucoup de variantes dans le typage.

Par exemple, ML tolère les fonctions récursives.
En introduisant une primitive Y tel que $\vdash Y : (\alpha \to \alpha) \to \alpha$.
[On perd : M typable $\Rightarrow$ M fortement normalisable]

Il existe aussi des systèmes de types bien plus expressifs.
On va le voir dans le chapitre suivant.